VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2017 Tomáš Biesok

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Biesok**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: AstrumQ Interactive, s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Pavel Dohnálek**

Konzultant bakalářské práce: Ing. Daniel Robenek

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 19.dubna 2017

Bielek Tomáš

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 19.dubna 2017

Abstrakt

Tato práce pojednává o absolvování mé odborné praxe ve firmě AstrumQ Interactive s.r.o.. Cílem této práce je ve stručnosti představit firmu, v níž má odborná praxe probíhala. Dalším bodem je popsání zadávaných úkolů a jejich řešení. Konec této práce obsahuje zhodnocení získaných zkušenosti s práci v týmu nad reálnými projekty.

Klíčová slova: Android, GIT, mobilní aplikace, MVP, odborná praxe

Abstract

This thesis describes my practice in AstrumQ Interactive s.r.o. company. The aim of this thesis is to briefly introduce the company in which my professional practice took place. The next point is describe of assigned tasks and their solutions. The end of this work includes an evaulation acquired experiences on the real projects in teamwork.

Key Words: Android, GIT, mobile application, MVP, professional practice

Obsah

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
IDE	–	Integrated Development Environment
HTTP	–	Hypertext Transfer Protocol
JSON	–	JavaScript Object Notation
MVP	–	Model-View-Presenter
SDK	–	Software Development Kit
URL	–	Uniform Resource Locator
XML	–	Extensible Markup Language

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Absolvování odborné praxe jsem si zvolil z toho důvodu, protože jsem chtěl poznat, jak funguje práce na reálných projektech v týmu a získat ještě během studia nějaké zkušenosti. Moje praxe probíhala ve firmě AstrumQ Interactive s.r.o., kde jsem pracoval jako Android vývojář. Tuto pozici jsem si zvolil z toho důvodu, protože se chci do budoucna zabývat vývojem aplikací pro tuto platformu. Když jsem na této pozici začínal, měl jsem pouze základní znalosti v této oblasti.

V druhé kapitole ve stručnosti představím firmu, ve které jsem dostal příležitost svou odbornou praxi absolvovat, a také mé zařazení na pracovní pozici. Je zde ve stručnosti uvedena i náplň mé práce.

Třetí kapitola popisuje technologie, se kterými jsem během své odborné praxe pracoval a použil je vývoje jedné aplikace.

Čtvrtá a pátá kapitola je věnována seznámení se s úkoly, na kterých jsem pracoval, postupy jejich řešení a použité knihovny a technologie. Úkolu bylo však více, ale vybral jsem pouze ty, které dle mého názoru byly zajímavé.

Poslední kapitoly jsou věnovány shrnutí mých zkušeností, které jsem měl na začátku a následně na konci praxe. Je zde také celkový závěr a mé zhodnocení.

2 Představení firmy

2.1 O firmě

AstrumQ Interactive s.r.o. je mladou firmou sídlící v Ostravě od roku 2012, zabývající se vývojem mobilních a webových aplikací. Během své krátké existence získala již několik úspěchů. Spolupracuje s klienty malých a středních firem z mnoha odvětví. Vývoj je zaměřen především do oblastí obchodu, vzdělávání, gastronomie, zábavy a marketingu. V rámci spolupráce nabízí servis mobilních a webových aplikací a odborné poradenství. Firma pracuje jak na malých projektech, tak i na větších. Mezi větší projekty patří například bitcoinová peněženka BitOasis, nebo také webový portál Sportnect.com, který slouží k organizovaní sportovních aktivit pro veřejnost.[1]

2.2 Pracovní zařazení

Do firmy jsem nastoupil jako Android vývojář. Po nastoupení jsem dostal jako první testovací projekt vytvořit aplikaci, která sloužila jako jednoduchý chat, který komunikoval s webovým API. Projekt sloužil k ověření mých znalostí s vývojem na Android, které jsem v té době již měl. Po dokončení této aplikace byla předána na code review zkušenému programátorovi, který zhodnotil můj kód a dal mi zpětnou vazbu, na čem bych měl zapracovat nebo jaké knihovny bych měl začít používat.

Následně jsem se seznamoval s různými Java knihovnami (například pro síťovou komunikaci, asynchronní stahování obrázků z internetu, atd.) a testoval webové API pro portál Sportnect.com. Mezitím jsem měl možnost se s touto webovou aplikaci také blíže seznámit.

Nakonec jsem se podílel na tvorbě aplikace Sportuj v Ostravě, která čerpala data ze Sportnectu a sloužila pro organizování sportovních událostí pouze pro statutární město Ostravu. Tuto aplikaci jsem tvořil již se zkušeným vývojářem a během tvorby jsem se naučil používat verzovací systém GIT a pracovat v týmu.

3 Použité technologie

3.1 Java

Jedná se o objektově orientovaný programovací jazyk vyvinut společností Sun Microsystems, představen v roce 1995. Patří mezi jeden z nejpoužívanějších programovacích jazyků na světě. Původně měl být využit pro spotřební elektroniku, ale rozšířil se i do dalších systémů. Java je interpretovaný jazyk - to znamená, že se kód nepřekládá přímo do strojového kódu, ale do tzv. Java bajtkódu, který je následně interpretován virtuálním strojem Java. Z tohoto důvodu se na platformě jedná o nezávislý jazyk. O správu paměti se stará tzv. Garbage Collector, který automaticky maže objekty, které se již nepoužívají.[2]

3.2 Android SDK

Jedná se o balíček vývojářských nástrojů, které zahrnují debugger, knihovny, emulátor zařízení a spoustu dalších. V současné době mezi podporované vývojářské platformy patří moderní distribuce Linuxu, Windows 7 nebo novější a Mac OS X 10.5.8 nebo novější.

Do konce roku 2014 patřily mezi oficiálně podporované vývojové prostředí, například Eclipse s Android Development Tools pluginem nebo NetBeans IDE s pluginem podporující Android vývoj. Od roku 2015 je oficiálně podporováno vývojářské prostředí Android Studio, vyvinuto Googlem. Vývojáři nicméně mohou využívat stále jiné prostřední pro vývoj, nebo také mohou použít jakýkoliv textový editor pro editaci Java a XML souborů, a ty následně pomocí příslušných nástrojů skrz příkazový řádek sestavit a ladit.

Android SDK také podporuje starší verze Androidů v případě, že chce vývojář vyvíjet aplikaci i na starší zařízení.[3]

3.3 Google Volley

Google Volley je knihovna, která slouží k vytváření HTTP požadavků a k jednoduché a rychlé komunikaci přes síť. Nabízí pro vývojáře řadu výhod:

- automatické plánování HTTP požadavků
- více souběžných síťových spojení
- řeší cachování
- podpora priorit požadavků
- možnost zrušení jednoho požadavku nebo rovnou celého bloku
- a mnoho dalších.

Volley není vhodná pro stahování velkého objemu dat nebo pro streamování, protože všechny odpovědi drží během parsování v paměti.[4]

3.4 Gson

Gson je Java knihovna používaná ke konverzi Java objektů na JSON a naopak. K tomuto účelu poskytuje jednoduché metody `toJson()` a `fromJson()`. Podporuje i konverzi na vlastní objekty, které se definují jako třídy.[5]

3.5 Picasso

Picasso je jednoduchá knihovna používaná k asynchronnímu stahování obrázku z internetu nebo jeho načtení z jiných zdrojů. Obsahuje metody pro různé transformace s minimálním využitím paměti. Navíc obrázky umí automaticky cachovat jak do paměti, tak na disk.[6]

4 Seznam zadaných úkolů

4.1 Aplikace pro firemní chat

Mým prvním úkolem bylo vytvořit pomocí Javy a Android SDK aplikaci, která sloužila jako firemní chat. Jednalo o testovací projekt, který se zadával všem novým členům týmu k ověření jejich znalostí se systémem Android. Dostal jsem zadání úkolů, odkaz na wireframe aplikace, odkaz na webovou aplikaci a odkaz na API dokumentaci. Jelikož se jednalo o mou úplně první Android aplikaci, strávil jsem nad její implementaci zhruba 10 dnů. Seznam zajímavých úkolů:

- Připojování se k serveru
- Zprávy uživatelů
- Odeslání emailu uživateli

4.2 Aplikace Sportnect

Po vytvoření aplikace pro firemní chat jsem dostal za úkol otestovat API pro naši webovou aplikaci Sportnect, kterou vyvíjeli další vývojáři z týmu. Během tohoto úkolu jsem se naučil používat knihovny pro usnadnění vývoje Android aplikací, Best Practice, a také jsem se učil komunikace s ostatními členy týmu. Nad touto aplikaci a učením se jsem strávil kolem 20 dnů. Seznam zajímavých úkolů:

- Přihlašovací aktivita a MVP
- Komunikace se serverem a Google Volley
- Konverze JSON objektu na Java objekt a GSON

4.3 Aplikace Sportuj v Ostravě

Posledním větším úkolem, na kterém jsem pracoval, byl vývoj aplikace, která již bude vydaná pro veřejnost a jednalo se o aplikaci podobnou Sportnectu, která byla ale zaměřená pouze na statutární město Ostravu. Během tohoto zadání jsem spolupracoval na tomto projektu se zkušenějším vývojářem, kde jsme si již rozdělovali úkoly mezi sebe. Zde jsem se již učil reálné spolupráce s dalším členem týmu na jednom projektu. Na této aplikaci jsem strávil zbylé dny své odborné praxe ve firmě. Seznam zajímavých úkolů:

- Abstraktní třída pro zobrazování fragmentů
- Zpracování chybových odpovědí ze serveru

4.4 Dílčí úkoly

Během své praxe jsem se setkal i s jinou prací, než jen s programováním aplikací. V řešení úkolů se nachází krátký popis. Tyto úkoly jsem dostával průběžně během praxe.

5 Řešení zadaných úkolů

5.1 Aplikace pro firemní chat

V době, kdy jsem dostal tento testovací projekt, měl jsem v oblasti systému Androidu jen základní znalosti a teprve jsem se s tímto systémem začínal učit pracovat.

Nejdříve jsem vytvořil přihlašovací aktivitu. Aktivita je obrazovka, která uživateli zobrazuje grafické rozhraní a slouží k interakci s ním. V rozvržení prvků byly pouze dva prvky třídy `EditText` pro vyplnění emailu a hesla, `CheckBox` pro trvalé přihlášení a dvě tlačítka, jedno pro přihlášení a druhé pro registraci. Jelikož po grafické stránce nebyly na tuto aktivitu kladeny žádné nároky, pustil jsem se do implementace funkčnosti přihlášení uživatele.

5.1.1 Připojování se k serveru

Existuje také webová verze tohoto projektu, se kterým moje aplikace spolupracovala. Pro tento účel bylo vytvořeno a zdokumentováno webové API. Skrze toto rozhraní lze přistupovat k datům, jako jsou například zprávy v chatu, informace o uživatelích, profilový obrázek uživatele, atd.

Nejdřív bylo třeba ověřit údaje uživatele, který se chtěl přihlásit do aplikace. Uživatel vyplní do vstupů svůj email a heslo, a poté klikne na tlačítko pro přihlášeni. Data jsou odeslány na server, který je zpracuje a vrátí zpět odpověď v JSON formátu. Jedná se o textový řetězec dat, který má svou strukturu a je nezávislý na platformě.

Vytvořil jsem třídu, která měla na starost připojení a komunikaci se serverem skrze protokol HTTP. K tomuto účelu jsem použil Java třídu `HttpURLConnection`, která slouží k odesílaní požadavků na HTTP server.[7] Požadavky odesílané na server používají v mém případě dvě metody (GET a POST). Metoda GET slouží k získání dat ze serveru a patří mezi nejpoužívanější ze všech metod. Do požadavku lze umístit parametry, které se zapisují přímo za URL. Další metodou používanou v tomto projektu byla metoda POST. Tato metoda slouží k odesílaní uživatelských dat na server. Používá se například k odeslání formuláře na webu. Data na server může odeslat i metoda GET, ale ve většině případů se k tomuto účelu používá právě metoda POST, hlavně z důvodu většího objemu dat, která tato metoda může pojmout a z důvodu vyšší bezpečnosti, pokud nechceme nějaká data zobrazovat jako součást URL (například hesla a jiná citlivá data).[8] V mé třídě, která sloužila ke komunikaci se serverem, jsem implementoval metody pro:

- nastavení URL
- nastavení HTTP metody
- nastavení parametrů požadavku
- nastavení hlavičkových parametrů
- odeslaní požadavku

- získání kódu odpovědi
- získání odpovědi serveru jako řetězec
- získání odpovědi serveru jako JSON objekt
- odpojení se od serveru

Třída fungovala tím způsobem, že nejdříve se nastavily její atributy, jako jsou URL, HTTP metoda, parametry požadavku (pouze u POST metody) a hlavičkové parametry. Parametry pro metodu GET se uváděly přímo za URL. Následně se zavolala metoda pro odeslání požadavku na server.

Ta z nastavených atributů třídy sestavila spojení se serverem a požadavek na něj odeslala. Po přijetí odpovědi ze serveru nejdříve získala kód odpovědi, jestli požadavek proběhl v pořádku, a podle něj vytvořila vstupní datový proud (buď klasický nebo chybový). Následně byla zavolána funkce pro získání odpovědi jako řetězec nebo jako JSON objekt. Ta byla dále programem rozparsována a zpracována.

Pro přihlášení do aplikace, server vracel při úspěchu ID uživatele a token a při neúspěchu kód chyby a zprávu. Token se používal pro autorizaci během odesílání jiných požadavků na server, které jej vyžadovaly. Vkládal se do hlavičky požadavku. Abych se ale nemusel před každým dalším požadavkem, který autorizaci vyžadoval, doptávat na token, vytvořil jsem třídu `LoggedUser`, která měla atributy `ID` jako celočíselný typ a `token` jako řetězec, a do této třídy, přes parametry konstruktoru, jsem tyto údaje uložil. Dále jsem této třídě vytvořil metody pro získání těchto údajů a metodu na jejich vynulování. (Výpis 1.)

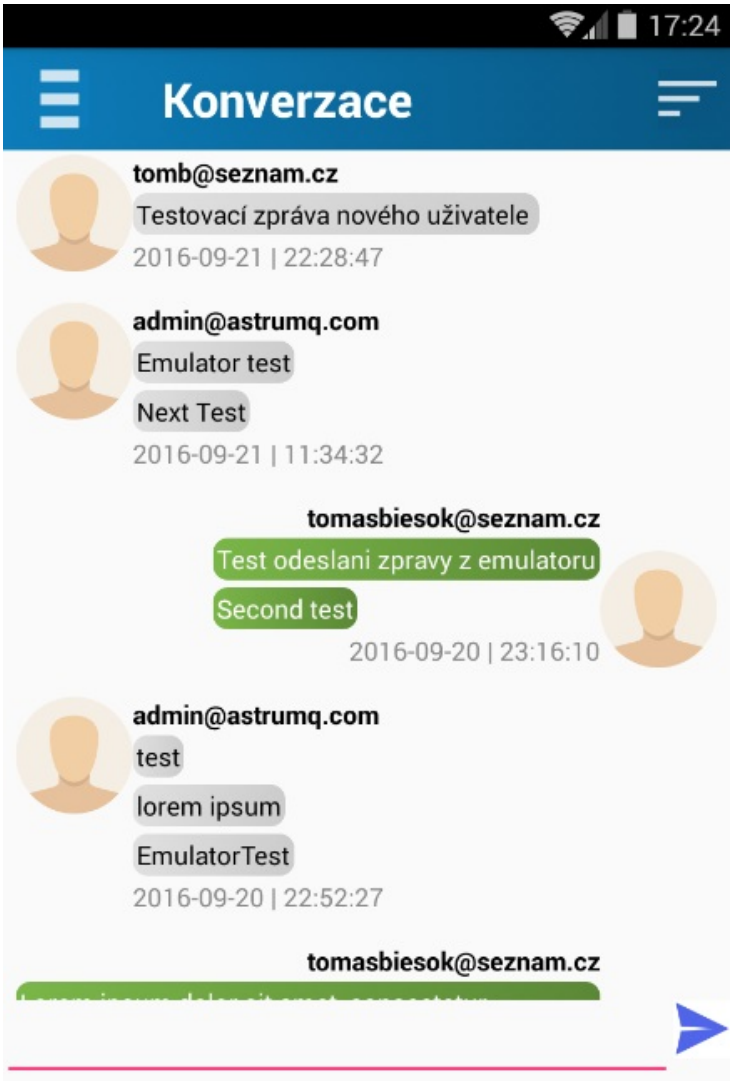
```java
public class LoggedUser {
    private int id;
    private String token;

    public LoggedUser(int id, String token) {
        this.id = id;
        this.token = token;
    }
    public int getId() {
        return id;
    }
    public String getToken() {
        return token;
    }
    public void clearData() {
        id = 0;
        token = "";
    }
}
```

Výpis 1: Třída LoggedUser pro uložení ID a tokenu přihlášeného uživatele

5.1.2 Zprávy uživatelů

Dalším úkolem bylo získat ze serveru zprávy uživatelů a vypsat je na displej. K tomuto účelu jsem vytvořil novou aktivitu, která se zobrazila po tom, co se uživatel úspěšně přihlásil. Data se stahovala stejně, jako ID a token během přihlášení uživatele.

Zprávy byly graficky rozlišeny podle toho, kdo zprávu odeslal. Zprávy přihlášeného uživatele měly zelené pozadí a byly zarovnány k pravému okraji displeje a zprávy ostatních měly šedé pozadí a byly zarovnány k levému okraji displeje. (Obrázek 1.) Pro zobrazení jedné zprávy (bubliny) jsem vytvořil třídu `MessageBubble`, která dědila ze třídy `TextView`. Třída `TextView` slouží k zobrazení textu v layoutu. K mé vytvořené třídě, pro zobrazování bublin, jsem vytvořil ve složce `Drawable` dva soubory, ve kterých byl definován vzhled prvku. Jeden soubor byl pro zprávy přihlášeného uživatele a druhý pro ostatní uživatele. (Výpis 2.) Třída `MessageBubble` obsahovala atribut `isMyMessage` typu boolean, na jejíž hodnotě záviselo, jak se bublina se zprávou vykreslí (jestli patří přihlášenému uživateli, nebo jinému).



Obrázek 1: Aktivita pro zobrazení chatu

```xml
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">

    <corners android:radius="8dp" />

    <gradient
        android:startColor="@color/message_my_start"
        android:endColor="@color/message_my_end" />

    <padding
        android:bottom="2dp"
        android:left="2dp"
        android:right="2dp"
        android:top="2dp" />

</shape>
```

Výpis 2: Vzhled bubliny se zprávou pro přihlášeného uživatele

Dalším atributem třídy byl řetězec, ve kterém byl uložen obsah celé zprávy. Pokud text přesáhl maximální délku počtu zobrazovaných znaků, celý se uložil do atributu `fullMessage`, ale v bublině byl ořezán jen na určený počet znaků. Tento počet byl definován v konstantním atributu `MAX_MESSAGE_LENGTH_IN_BUBBLE`. Ve třídě jsem vytvořil metodu `setBubbleText()` s parametrem text typu `String`, která měla za úkol ořezat text, pokud byl dlouhý, a zobrazit jej v prvku. (Výpis 3.)

```java
public void setBubbleText(String text) {
    if (text.length() > MAX_MESSAGE_LENGTH_IN_BUBBLE) {
        fullMessage = text;
        setText(text.substring(0, MAX_MESSAGE_LENGTH_IN_BUBBLE - 1) + "...");
    } else {
        setText(text);
    }
}
```

Výpis 3: Implementace metody setBubbleText pro ořezání dlouhého textu

S dlouhými zprávami nastal problém; bylo je třeba zobrazit celé, pokud byly v bublině ořezány. K tomuto účelu jsem vytvořil novou aktivitu, ve které se vykreslovala fotografie uživatele, kterému zpráva patřila; jeho email, jméno a příjmení, datum odeslání zprávy a celý text. Aktivitu jsem rozdělil na horní a spodní část. V horní části byla zobrazena profilová fotka uživatele

a pod ní jeho email. Spodní část obsahovala `ScrollView`, který slouží ke skrolování obsahu, pokud se celý nevejde na displej. V něm bylo vyplněno jméno a příjmení uživatele, datum zprávy a celý text. (Obrázek 2.) Aby se uživatel dostal do této aktivity, musel kliknout na bublinu se zprávou, kterou chtěl zobrazit celou, pokud byla ořezána. Při kliknutí na bublinu se zavolala metoda `onClick()`, která měla za úkol vytvořit instanci objektu `Intent`, ve kterém se předaly data ve formě klíč - hodnota a ty se následně v nové aktivitě v metodě `onCreate()` zpracovaly.

Intenty jsou jednoduché objekty, které slouží jako "most"mezi komponentami, jako je například aktivita, broadcast receiver nebo services. Lze do něj dostat primitivní data (int, float, boolean, String, Serializable,...) a ty pak zpracovat, například v jiné aktivitě. Existují dva druhy intentů: explicitní a implicitní. Explicitní intenty přímo vědí, co chtějí provést (jakou třídu chtějí spustit). Implicitní intenty pouze popisují záměr, který chtějí provést, ale způsob provedení již zajistí sám systém, a ten zjistí, které aktivity umějí daný intent zpracovat. Pokud se takových aktivit najde více, zobrazí se uživateli dialogové okno s možnostmi, ze kterých si uživatel vybere, kde chce daný intent zpracovat.[9]

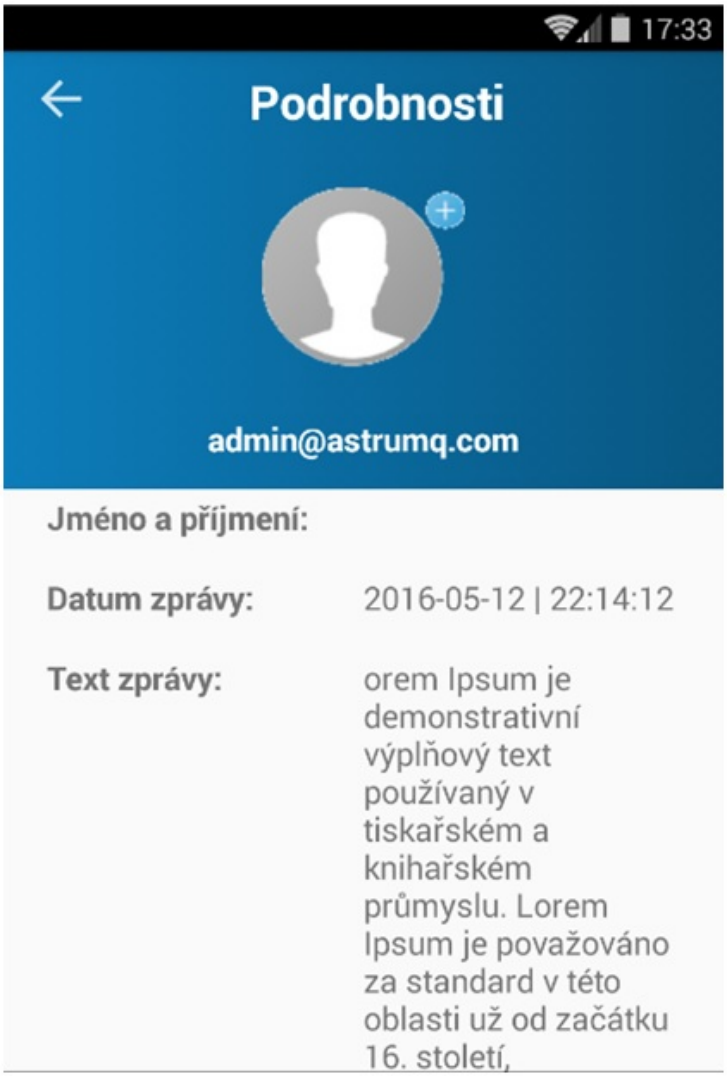
5.1.3 Odeslání emailu uživateli

V zadání jsem dostal úkol z aktivity podrobností uživatele implementovat možnost napsání tomuto uživateli email. K tomuto úkonu došlo po kliknutí na email pod profilovou fotografii. Tento problém jsem řešil pomocí implicitního intentu.

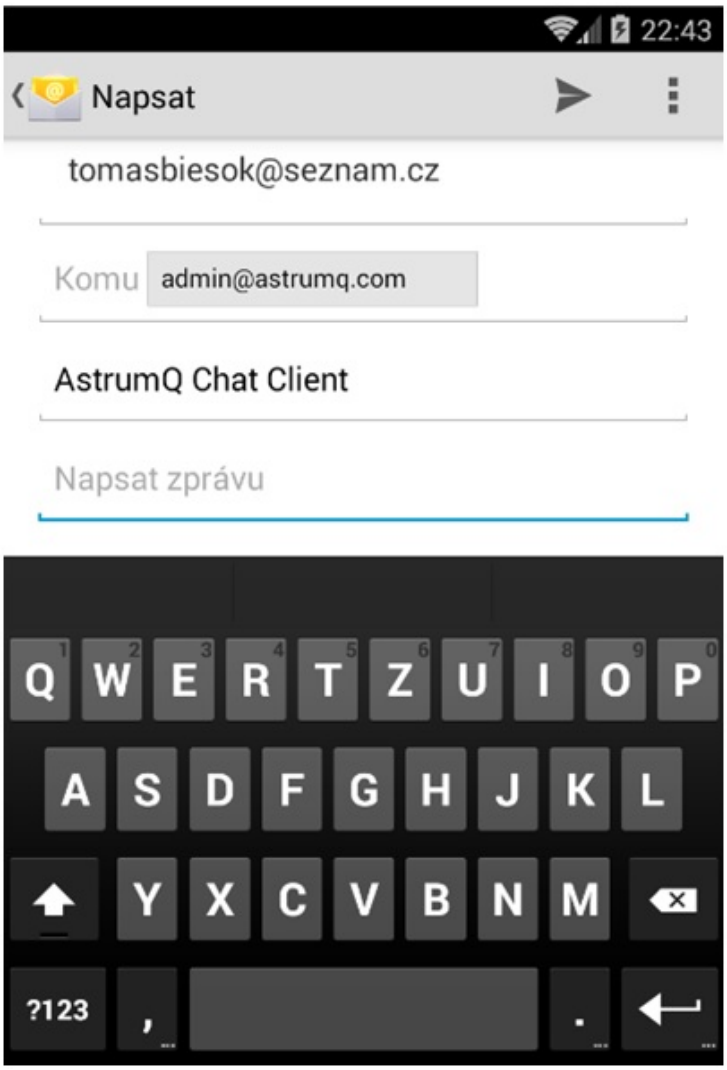
Vytvořil jsem metodu `sendEmail()`, která byla volána po kliknutí na prvek `TextView`, který zobrazoval email uživatele. V metodě jsem vytvořil instanci třídy `Intent`, a jako parametry konstruktoru jsem předal akci, která se provedla a URI objekt, který obsahoval příkaz na odeslání emailu na zadanou adresu. Akce intentů jsou statické a typu `String`. Vybral jsem akci `Intent.ACTION_SENDTO`, která odesílá zprávu specifikovanou podle typu dat.[10] Pomocí metody `putExtra()` jsem ještě do intentu vložil předmět emailu. Nakonec jsem zavolal metodu `startActivity()`, které jsem předal instanci intentu, a na ní ještě zavolal metodu `createChooser()`, která dá uživateli na výběr z aktivit, které umí daný záměr zpracovat. (Výpis 4.) Jelikož jsem na testovacím zařízení měl pouze jednu takovou aktivitu, žádný výběr se mi nezobrazil a aktivita pro zpracování záměru se spustila automaticky. (Obrázek 3.)

```
public void sendEmail(View v) {
    Intent intent = new Intent(
        Intent.ACTION_SENDTO,
        Uri.parse("mailto:" + String.valueOf(userEmail.getText())));
    intent.putExtra(Intent.EXTRA_SUBJECT, emailSubject);
    startActivity(Intent.createChooser(intent, ""));
}
```

Výpis 4: Implementace metody pro spuštění vhodné aktivity pro napsání emailu uživateli



Obrázek 2: Aktivita s podrobnostmi zprávy



Obrázek 3: Emailový klient pro odeslání emailu

5.2 Aplikace Sportnect

Po získání zpětné vazby na aplikaci, kterou jsem programoval pro ověření mých dosavadních znalostí, jsem dostal za úkol naučit se pracovat s několika Java knihovnami pro ušetření času a usnadnění práce během vyvíjení Android aplikací. Měl jsem také k dispozici dokumentaci k API pro náš webový portál Sportnect.com. Aby nedošlo k manipulaci s daty na ostrém serveru, ke kterým měla přístup i veřejnost, přistupoval jsem pouze k development serveru, který byl určen pro vývoj a testování.

5.2.1 Přihlašovací aktivita a MVP

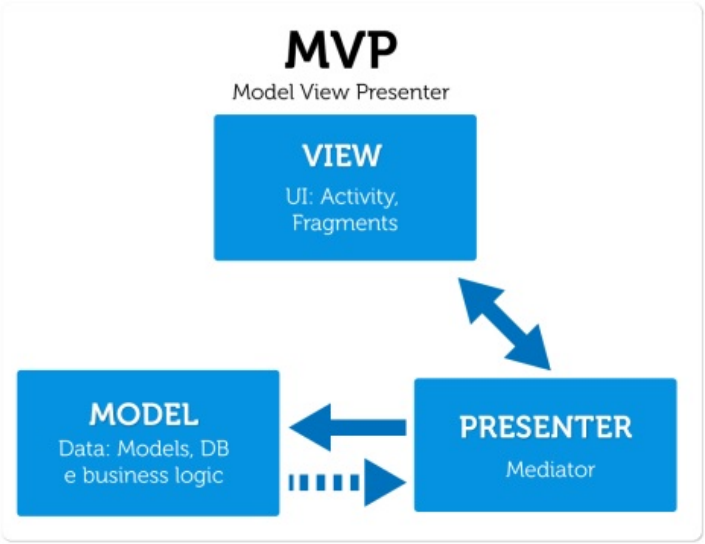
Stejně jako v předchozím projektu, i tady jsem potřeboval vytvořit přihlašovací aktivitu, aby bylo jasné, jaký uživatel s aplikací pracuje. Jelikož jsem se snažil svůj kód nějak zpřehlednit, hledal jsem na internetu rady a návody, jak toho dosáhnout. Narazil jsem na architektonický vzor Model-View-Presenter.

Model-View-Presenter (MVP) vychází ze vzoru Model-View-Controller. Používá se k oddělení kódu do tří vrstev (komponent), aby byl kód přehlednější. Vrstvy jsou následující:

- **model** - slouží pro přístup k datům a manipulace s nimi (např. data z databáze, sítě,...)
- **view** - data, která získal od modelu, převádí do podoby, kterou následně zobrazí uživateli
- **presenter** - vrstva, která propojuje model a view a slouží ke zpracovávání událostí skrz view (např. kliknutí na tlačítko) a manipulaci s daty skrz model (např. stažení dat ze serveru)

V mém případě do vrstvy view spadaly aktivity a fragmenty, do vrstvy model síťová komunikace přes API se serverem a vrstva presenteru reagovala na události uživatele. Na jejich základě vysílala požadavky modelu, který vracel data zpět presenteru a ten nad vrstvou view volal metody, do kterých tyto data předával a ty byly následně zobrazovány na displeji.

Rozhodl jsem se, že vytvořím package s názvem `mvp`, do které jsem vytvořil další balíčky, které se již staraly pouze o konkrétní fragmenty a aktivity. Pro logovací aktivitu jsem vytvořil package `loginactivity`, do které jsem vytvořil interface `LoginMVP`. Ten obsahoval další čtyři interfacy, ve kterých byly deklarovány metody pro všechny vrstvy. Vrstva presenteru obsahovala dva interfacy. Jeden byl pro komunikaci view s presenterem, druhý byl pro komunikaci modelu s presenterem. (Výpis 5.)



Obrázek 4: Vztahy mezi vrstvami MVP[11]

```
public interface LoginMVP {
    interface RequiredViewOps {
        void onSuccess();
        void onError(String message);
    }

    interface PresenterOps {
        void loginToApp(String email, String password);
    }

    interface RequiredPresenterOps {
        void onSuccessfulResponse(JSONObject response);
        void onError(String message);
    }

    interface ModelOps {
        void loginToApp(String email, String password);
        void mapErrorResponse(JSONObject response);
        void mapLoggedUser(JSONObject response);
    }
}
```

Výpis 5: Implementace interfacu pro MVP LoginActivity

Následně jsem tyto interfacy přiřadil třídám, ke kterým měly patřit. `RequiredViewOps` patřily pro vrstvu view, kterou v mém případě byla přihlašovací aktivita. `PresenterOps` a `RequiredPresenterOps` patřily třídě `LoginPresenter`, kterou jsem vytvořil a neměla žádného předka. `ModelOPS` patřil třídě `LoginModel`, která obsluhovala instanci třídy `ApiClient` na komunikaci ze serverem.

Mechanismus fungoval tak, že uživatel vyplnil email a heslo a po kliknutí na tlačítko "Přihlásit se"se spustila metoda presenteru `loginToApp()`, kde se v parametrech předal email a heslo ze vstupů. Presenter pouze zavolal metodu modelu `loginToApp()` a stejné parametry předal dál. Metoda modelu vytvořila instanci `HashMap`, kterou jsem naplnil parametry, které se odesílají na server v požadavku. Následně metoda získala instanci třídy `ApiClient` a pomocí ní odeslala požadavek na server. Třída modelu měla také implementované interfacy pro příjem úspěšné i neúspěšné odpovědi ze serveru. V těchto metodách volala metody druhého interfacu presenteru; buď metodu `onSuccessfulResponse()`, ve které předávala pomocí parametru úspěšně přijatý JSON objekt ze serveru, nebo metodu `onError()`, kde předávala chybovou zprávu. Pokud přišla chybová zpráva, zobrazila se jako Toast na displeji. Pokud přišel ze serveru JSON objekt, musel presenter ještě ověřit, zda přišel objekt s chybovou zprávou nebo s úspěšným přihlášením uživatele. To ověřoval podle toho, zda obsahoval atribut code, ve kterém byl uložen kód chyby. Objekt s chybovou zprávou vracel v případě, že uživatel zadal špatné údaje, uživatel neexistoval, apod. Pokud atribute code neobsahoval, jednalo se o úspěšné přihlášení. Oba JSON objekty byly v modelu mapovány na doménové objekty. Pokud přihlášení proběhlo v pořádku, spustila se následující aktivita, která již zobrazovala konkrétní data ze serveru, v opačném případě se zobrazila chybová zpráva jako Toast. (Obrázek 5.)

5.2.2 Komunikace se serverem a Google Volley

Stejně jako u aplikace pro firemní chat jsem u této aplikace potřeboval komunikovat se serverem. K tomuto účelu mi bylo doporučeno pár knihoven, ale nejvíce z nich se mi zalíbila knihovna Google Volley. Ta slouží k vytváření a zpracovávání HTTP požadavků. Navíc požadavky zpracovává asynchronně, tudíž vývojář nemusí řešit vlákna a mezivláknovou komunikaci.

Vytvořil jsem třídu `ApiClient`, která používala návrhový vzor jedináčka (Singleton). Tento návrhový vzor vytvoří pouze jednu instanci této třídy, kterou následně používá v celé aplikaci. Třída v sobě držela privátní statickou proměnnou `instance`, měla pouze privátní bezparametrový konstruktor a veřejnou statickou metodu `getInstance()`, která vracela instanci třídy. Pokud ale instance neexistovala, metoda jí vytvořila. (Výpis 6.)

Obrázek 5: LoginActivity pro Sportnect

```
public static ApiClient getInstance() {
   if (instance == null) {
      instance = new ApiClient();
   }
   return instance;
}
```

Výpis 6: Statická metoda pro získání instance jedináčka

Ve třídě jsem vytvořil metodu `sendRequest()`, kterou jsem přetížil pro různé parametry metody. Hlavní parametry, které byly u každého požadavku vyžadovány, byl tzv. "API Endpoint", rozhraní pro úspěšnou odpověď ze strany serveru a chybové rozhraní. API Endpoint byl typu `String` a obsahoval koncovou cestu, kde se požadavek odesílal. Všechny přetížené metody vždycky volaly tu, která obsahovala všechny parametry. Za nedosazené parametry se vkládaly vždycky výchozí hodnoty nebo prázdné instance. Pokaždé se vytvořila instance třídy `JsonObjectRequest()`, která reprezentovala jeden požadavek. V parametrech konstruktoru se předávala HTTP metoda, URL pro endpoint, parametry pro dotaz (pouze u metody POST), rozhraní pro úspěšnou odpověď serveru a rozhraní pro chybu. Pokud se jednalo o metodu GET, parametry dotazu byly předány jako součást URL. Proto bylo v metodě třeba rozlišovat, zda se jedná o požadavek pro metodu GET, nebo POST. Třída `Volley` měla statickou metodu `newRequestQueue()`, která vytvořila novou frontu požadavků a následně do této fronty se pomocí metody `add()` přidal požadavek, který byl následně zpracován. (Výpis 7.)

Aby bylo možné zpracovat odpověď serveru, z tohoto důvodu byly předány rozhraní v metodě `sendRequest()`. Odpověď byla vrácena do metod těchto rozhraní, které většinou implementovala třída, která daný požadavek odesílala. V mém případě tyto rozhraní vždy implementovala vrstva modelu dle architektury MVP. V této vrstvě již byly odpovědi zpracovávány.

```
public void sendRequest(String endpoint, JSONObject jsonRequest, Response.
    Listener onResponseListener, Response.ErrorListener onErrorListener) {
   JsonObjectRequest request;
   switch (getEndpointMethod(endpoint)) {
      case JsonObjectRequest.Method.POST:
         request = new JsonObjectRequest(
            getEndpointMethod(endpoint),
            getUrlForEndpoint(endpoint),
            jsonRequest,
            onResponseListener,
            onErrorListener);
      break;

      default:
         request = new JsonObjectRequest(
            getEndpointMethod(endpoint),
            getUrlForEndpoint(endpoint) + getMethodGetParameters(jsonRequest),
            jsonRequest,
            onResponseListener,
            onErrorListener);
      break;
   }
   Volley.newRequestQueue(SportnectApp.getContext()).add(request);
}
```

Výpis 7: Implementace hlavní metody pro odeslání požadavku

5.2.3 Konverze JSON objektu na Java objekt a GSON

Server vracel odpověď ve formátu JSON. Pro práci s přijatými daty je bylo potřeba převést na Java objekt. Java obsahuje knihovny pro zpracování JSON formátu, ale bylo by třeba pomocí nich napsat vlastní parser. Proto jsem pro tento účel použil knihovnu GSON, která se o konverzi postará.

Nejdříve jsem vytvořil doménové objekty, jejichž atributy měly stejné názvy, jako byly názvy klíčů v JSON odpovědi. Pokud by se názvy neshodovaly, interpret by nevěděl, do kterého atributu má jakou hodnotu přiřadit. Občas nastal problém, když jsem chtěl mít ve svém doménovém objektu atribut pojmenován jinak, než byl v JSONu. K tomuto účelu sloužila anotace `@SerializedName()`, která jako parametr typu `String` přijímala skutečný název klíče JSON objektu, ale název samotného atributu v Java objektu jsem mohl už zvolit libovolný. Navíc jsem

si vytvořil třídu `ApiConstants`, do které jsem ukládal jako veřejné finální statické atributy názvy klíčů JSON objektů a ty následně používal ve zmiňovaných anotacích. Příklad doménového objektu pro `AccessToken` je v následujícím výpisu. (Výpis 8.)

```
public class AccessToken {
    @SerializedName(ApiConstants.PARAMETER_ACCESS_TOKEN)
    public String accessToken;

    @SerializedName(ApiConstants.PARAMETER_TOKEN_TYPE)
    public String tokenType;

    @SerializedName(ApiConstants.PARAMETER_EXPIRES_IN)
    public int expiresIn;

    @SerializedName(ApiConstants.PARAMETER_REFRESH_TOKEN)
    public String refreshToken;

    @SerializedName(ApiConstants.PARAMETER_USER)
    public String userId;
}
```

Výpis 8: Model doménového objektu pro AccessToken

Když jsem připravil doménový objekt pro `AccessToken`, vytvořil jsem třídu `GsonMapper`, která byla jedináček a obsahovala instanci třídy `Gson`. Jelikož jsem potřeboval konvertovat JSON objekt na Java objekt, pracoval jsem u Gsonu pouze s metodou `fromJson()`, do které jsem předával v parametrech JSON objekt a třídu doménového objektu, na kterou se data měly namapovat (v tomto případě `AccessToken.class`). Občas se stalo, že server vrátil pole JSON objektů, které byly pojmenovány celými čísly. Tento problém jsem vyřešil tak, že jsem v `GsonMapper` vytvořil metodu `fromJsonAsList()`, ve které jsem předával JSON pole objektů ze serveru a třídu doménového objektu, na který se měly jednotlivé JSON objekty namapovat. Pole jsem procházel cyklem for, z něhož jsem získal jednotlivý objekt na základě celého čísla a ten jsem uložil do seznamu, který jsem po skončení cyklu vrátil. Abych zachoval obecnost metody a mohl jí použít i pro jiné doménové objekty, využil jsem zde genericitu. (Výpis 9.)

Takto vrácené objekty byly dále zpracovávaný programem.

```
public <T> List<T> fromJsonAsList(JSONArray response, Class<T> type) {
   List<T> list = new LinkedList<>();
   for (int i = 0; i < response.length(); i++) {
      list.add(fromJson(response.optJSONObject(i), type));
   }
   return list;
}
```

Výpis 9: Metoda pro získání seznamu objektů z pole odpovědi serveru

5.3 Aplikace Sportuj v Ostravě

Po tom, co jsem získal větší znalosti s vývojem aplikací na platformu Androidu, začal jsem pracovat na vývoji aplikace, která v budoucnu bude uvolněná pro veřejnost. Na tomto projektu jsem již pracoval s dalším členem týmu, jehož zkušenosti s programováním byly na vyšší úrovni, než ty moje. Učil jsem se zde používat další nové knihovny a naučil jsem se zde také celkem dobře pracovat s verzovacím systémem GIT.

5.3.1 Abstraktní třída pro zobrazování fragmentů

Jedním z problémů bylo, jakým způsobem budeme uživateli zobrazovat grafické rozhraní. Zvolili jsme tedy postup, kdy jsme vytvořili jednu aktivitu obsahující hlavní fragment, a v něm jsme zobrazovali již konkrétní fragmenty s grafickým rozhraním. Mým úkolem bylo připravit hlavní fragment.

Z grafického návrhu bylo patrné, že každá sekce bude obsahovat záložky, ve kterých se budou zobrazovat uživateli konkrétní fragmenty. Abych toho docílil, přidal jsem do třídy objekt `TabLayout`, který slouží pro zobrazení záložek. Potřeboval jsem ještě prvek na zobrazování konkrétních fragmentů. K tomu jsem využil objekt `ViewPager`, který slouží k zobrazování layoutů, kterými lze procházet pomocí gesta horizontálním posunutím prstu.

Navíc jsem vytvořil vlastní třídu `PagerAdapter`, která dědila z třídy `FragmentStatePagerAdapter`. Tato třída slouží ke spravování každé stránky fragmentu. Také se stará o obsluhu ukládání a obnovování stavu jednotlivých fragmentů.[12] `PagerAdapter` obsahoval `HashMapu`, kde klíče byly celá čísla a hodnotou byly konkrétní instance fragmentů, které se právě používaly. Hodnota klíče byla zároveň indexem fragmentu, kterému patřila. Byla zde přepsána metoda `getItem()`, která vracela z `HashMapy` fragmenty na základě klíče (indexu). Pokud v mapě nebyl nalezen fragment, který se měl zobrazit, byl vytvořen a do mapy uložen. Tím se ušetřila paměť, aby nebylo třeba vytvářet fragmenty, které uživatele vůbec nezajímají. Každý z fragmentů implementoval rozhraní `ILabeled`. Jednalo se o rozhraní, které poskytovalo metodu `getLabel()`, jejíž implementace vracela vždy jakýsi štítek, který se následně zobrazil jako záložka fragmentu.

Třída fungovala tak, že v přepsané metodě `onCreateView()` se vytvořila instance třídy `View`, na kterou se pomocí instance třídy `LayoutInflater` vložil již existující layout a objektům `ViewPager` a `TabLayout` se pomocí ID přiřadily jednotlivé komponenty layoutu. Toto vytvořené view bylo pak metodou vráceno. Další přepsaná metoda `onStart()` spustila funkci `initTabs()`, která zjistila, zda je počet fragmentů v adaptéru menší než dva, a pokud tuto podmínku splnila, instance třídy `TabLayout` záložky skryla. Následně pomocí metody `setTabTitles()` přiřadila záložkám štítky fragmentů. Nakonec instanci třídy `ViewPager` přiřadila adaptér, se kterým pracovala, a také propojila instanci třídy `TabLayout` s `ViewPagerem`.

5.3.2 Zpracování chybových odpovědi ze serveru

Jedním z problémů, který jsem také řešil, bylo zpracování chyby ze serveru. Ta mohla nastat v různých případech. Například pokud uživatel zadal v parametru neexistující ID sportu, server vrátil chybový JSON objekt, který obsahoval zprávu o neexistujícím sportu. Jelikož jsem na tomto projektu pracoval s dalším vývojářem, který používal jiné knihovny pro vývoj aplikací, musel jsem se s těmito knihovnami také seznámit. V tomto úkolu jsem se musel seznámit s knihovnou RxJava, která se používá pro reaktivní programování.

Reaktivní programování je asynchronní paradigma programování, zaměřené na datové toky a šíření změn. To znamená, že je možné vyjádřit jak statické, tak dynamické datové toky v programovacích jazycích jednoduše a provedení změny bude automaticky aplikováno prostřednictvím datového toku.[13]

Abych mohl chyby ze serveru zpracovávat, vytvořil jsem třídu `ApiErrorObserver`, která byla generického typu a dědila z abstraktní třídy `DisposableSingleObserver`, která patřila knihovně RxJava. V mé třídě jsem přepsal rodičovskou metodu `onError()`, která jako parametr obsahovala instanci třídy `Throwable`. Tato metoda byla vždy volána, pokud nastala chyba. V metodě jsem se dotazoval, zda se jedná také o instanci třídy `HttpException`, a pokud ano, tak jsem instanci na tuto třídu přetypoval a získal z ní tělo odpovědi. Přetížil jsem také metodu `onError()` a jako parametr jsem předával již instanci mé třídy `ApiError`. Tato třída byla doménový objekt, na který se mapoval příchozí JSON objekt ze serveru, který obsahoval chybu. Přetížená metoda byla abstraktní, tudíž ji bylo třeba implementovat v potomcích. V každém potomku pak bylo možno zpracovat chybovou zprávu individuálně. Rozhraní třídy `DisposableSingleObserver` navíc obsahovala abstraktní metodu `onSuccess()`, kde v parametrech se předával generický typ s odpovědi serveru. Implementace této metody byla také individuální na potomcích.

5.4 Dílčí úkoly

Během své praxe jsem kromě vývoje dostal i jiné úkoly. Hodně času jsem strávil studováním různých knihoven, používaných ve vývoji Android aplikací, ale měl jsem také možnost se několikrát podílet na testování webové aplikace Sportnectu a později i nově tvořené webové verzi Sportuj v Ostravě. Také jsem sepisoval dokument ohledně Best Practice vývoje na Android, který do budoucna firma využije jako pravidla, kterými se budou řídit Android vývojáři.

6 Závěr

6.1 Teoretické a praktické znalosti uplatněné v průběhu praxe

Během vývoje Android aplikací jsem využil především znalosti ze školy z různých předmětů. Z obou semestrů prvního ročníku jsem využil hlavně teoretické znalosti z programovacích předmětů jako byly *Algoritmy I*, *Algoritmy II*, *Programování I* a *Programování II*. Praktičtější věci jsem využil z předmětu *Programovací jazyky I*, který jsem absolvoval v zimním semestru druhého ročníku. Předmět byl zaměřen na programovací jazyk Java, ve kterém jsem vyvíjel aplikace během praxe. V zimním semestru třetího ročníku jsem měl předmět *Tvorba aplikací pro mobilní zařízení II*, ve kterém jsme se již zaměřili na platformu Androidu. Z tohoto předmětu jsem využil znalosti až po pár týdnech praxe.

6.2 Teoretické a praktické znalosti scházející v průběhu praxe

Na samotném začátku praxe mi scházely znalosti platformy Androidu. Základy jsem se učil během vývoje první aplikace a v předmětu *Tvorba aplikací pro mobilní zařízení II*. Dalším nedostatkem byla neznalost různých knihoven, používaných během vývoje aplikací ve firmách. Pracovat s těmito knihovnami jsem se učil během tvorby druhé aplikace. U poslední aplikace jsem se naučil základy práce s verzovacím systémem GIT. Dnes si již nedovedu představit tvorbu většího projektu v týmu s dalšími vývojáři bez tohoto nástroje.

6.3 Dosažené výsledky v průběhu praxe a jejich zhodnocení

V průběhu praxe jsem se naučil mnoho věcí. Naučil jsem se vyvíjet aplikace na platformu Androidu, zdokonalil jsem si znalosti objektově orientovaného programování v Javě, naučil jsem se pracovat s nejvyužívanějšími Java knihovnami, s verzovacím systémem GIT a spolupracoval jsem v týmu s lidmi na společném projektu. Měl jsem zde také možnost poznat různé technologie, které využívali jiní zkušení vývojáři. Také jsem se učil odhadovat čas, který strávím nad řešením jednotlivých úkolů.

Celkově z absolvování odborné praxe mám dobrý pocit a vůbec této volby nelituji. Získal jsem zde plno zkušeností, které určitě v budoucím povolání využiji a poznal jsem, jak funguje taková práce v reálném vícečlenném týmu na reálných projektech. Pro mne to byla dobrá příprava na budoucnost.

Literatura

[1] Vize. AstrumQ Interactive [online] [cit. 2017-02-13]
Dostupné z: https://astrumq.com/astrumq

[2] Java (Programovací jazyk). Wikipedie - Otevřená encyklopedie [online] [cit. 2017-03-05]
Dostupné z: https://cs.wikipedia.org/wiki/Java_(programovac%ED_jazyk)

[3] Android software development. Wikipedia - The Free Encyklopedia [online] [cit. 2017-03-05]
Dostupné z: https://en.wikipedia.org/wiki/Android_software_development

[4] Transmitting Network Data Using Volley. Android developers [online] [cit. 2017-03-05]
Dostupné z: https://developer.android.com/training/volley/index.html

[5] Google/gson. GitHub [online] [cit. 2017-03-05]
Dostupné z: https://github.com/google/gson

[6] Picasso [online] [cit. 2017-03-05]
Dostupné z: http://square.github.io/picasso

[7] HttpURLConnection (Java Platform SE 7) [online] [cit. 2017-03-15]
Dostupné z: http://docs.oracle.com/javase/7/docs/api/java/net/HttpURLConnection.html

[8] Hypertext Transfer Protocol - Dotazovací metody [online] [cit. 2017-03-15]
Dostupné z: https://cs.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[9] LACKO, Ľuboslav. Vývoj aplikací pro Android. Brno: Computer Press, 2015. ISBN 978-80-251-4347-6.

[10] Intent | Android Developers [online] [cit. 2017-03-18]
Dostupné z: https://developer.android.com/reference/android/content/Intent.html

[11] Model View Presenter MVP in Android, part 1 [online] [cit. 2017-03-19]
Dostupné z: http://www.tinmegali.com/en/model-view-presenter-android-part-1

[12] FragmentStatePagerAdapter | Android Developers [online] [cit. 2017-03-25]
Dostupné z: https://developer.android.com/reference/android/support/v4/app/FragmentStatePagerAdapter.html

[13] Reactive programming - Wikipedia [online] [cit. 2017-04-09]
Dostupné z: https://en.wikipedia.org/wiki/Reactive_programming